Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim

Farzad Farshchi
University of Kansas
farshchi@ku.edu

Qijing Huang
University of California, Berkeley
qijing.huang@berkeley.edu

Heechul Yun
University of Kansas
heechul.yun@ku.edu

Abstract

NVDLA is an open-source deep neural network (DNN) accelerator which has received a lot of attention by the community since its introduction by Nvidia. It is a full-featured hardware IP and can serve as a good reference for conducting research and development of SoCs with integrated accelerators. However, an expensive FPGA board is required to do experiments with this IP in a real SoC. Moreover, since NVDLA is clocked at a lower frequency on an FPGA, it would be hard to do accurate performance analysis with such a setup. To overcome these limitations, we integrate NVDLA into a real RISC-V SoC on the Amazon cloud FPGA using FireSim, a cycle-exact FPGA-accelerated simulator. We then evaluate the performance of NVDLA by running YOLOv3 object-detection algorithm. Our results show that NVDLA can sustain 7.5 fps when running YOLOv3. We further analyze the performance by showing that sharing the last-level cache with NVDLA can result in up to 1.56x speedup. We then identify that sharing the memory system with the accelerator can result in unpredictable execution time for the real-time tasks running on this platform. We believe this is an important issue that must be addressed in order for on-chip DNN accelerators to be incorporated in real-time embedded systems.

Keywords NVDLA, FireSim, RISC-V, FPGA, Cloud, DNN, Embedded Systems

1 Introduction

In recent years, deep neural networks (DNNs) are increasingly used in sophisticated embedded/robotics systems, such as self-driving cars and drones, due to their superior performance in solving complex perception and control tasks over traditional methods. However, DNNs are computationally intensive and have large memory requirement [7], while embedded systems generally impose strict constraints on power, size, and weight of their computing platforms.

To improve performance and efficiency of DNN processing, especially inference operations, in embedded systems, hardware DNN accelerators are being incorporated into embedded system-on-chips (SoCs). Recently, Nvidia introduced an open-source inference engine, called Nvidia Deep Learning Accelerator (NVDLA) [12], and integrated it into their Xavier SoC platform. They also announced a joint plan with Arm corporation to integrate NVDLA blocks in future ARM SoCs [13]. In August 2018, a semiconductor startup SiFive announced that they have integrated the NVDLA block into their open-source SoC platform [18]. In the demo presented at Hot Chips'18, NVDLA is implemented on an FPGA, which is connected to SiFive's Freedom U540 SoC platform[19] via an off-chip bus.[1]

We believe SiFive's integration of NVDLA is especially a useful platform for conducting research thanks to its open-source nature. However, the platform has several limitations: First, the FPGA board used to implement this platform costs about $7k [22]. This is an expensive piece of equipment for many research institutions and even some smaller companies. Second, a design implemented on FPGA has to be clocked at a lower frequency comparing to the equivalent ASIC implantation, however, DRAMs in both implementations are clocked at about the same frequency. Thus, the design implemented on FPGA sees relatively faster DRAM. Third, the current open-source release of SiFive's SoC platform does not support an L2 cache. In particular, the second and third limitations make performance-related research difficult on the platform.

To solve the aforementioned limitations, in this paper, we have integrated NVDLA into FireSim [10]. FireSim is an FPGA-accelerated full-system simulator, which runs on the Amazon cloud FPGAs. FireSim's target design is derived from the open-source RISC-V-based Rocket Chip SoC [4]. In FireSim, a special transform is applied on the target design that decouples it from the DRAM controller of the host FPGA and adds a memory model to the simulation environment. This allows us to model a realistic DRAM subsystem. We also model a last-level cache (LLC) similarly. Our FireSim integration solves the second and third limitations of SiFive's real FPGA-based integration. Moreover, since FireSim runs on the cloud, a research group may pay based on the hourly usage, without needing to purchase an expensive FPGA board. To compare with the cost of an FPGA board, the rate for on-demand access to the smallest Amazon FPGA instance is $1.65 per hour. Our integration of NVDLA with RISC-V SoC on FireSim is publicly available as open-source on GitHub.[2]

The rest of the paper is organized as follows. Section 2 describes the background on NVDLA and FireSim. Section 3 describes the NVDLA integration. In Section 4, we used NVDLA in FireSim to demonstrate how this platform can be used for performance analysis. We particularly focused on analyzing the performance achieved by sharing the LLC

EMC2'19, February 17, 2019, Washington D.C., USA
2019.

[1]The first author performed the FPGA integration for the demo during his internship at SiFive in summer 2018.
[2]https://github.com/CSL-KU/firesim-nvdla

with NVDLA and interference caused by sharing the memory system between the CPU cores and the NVDLA.

2 Background

In this section, we provide a brief background on NVDLA and FireSim.

2.1 NVDLA

NVDLA is an industry-grade open-source DNN inference engine, developed by Nvidia[12]. It is released as Verilog source code and is configurable at the build time to meet different performance, power, and area trade-offs. NVDLA mainly targets embedded systems and IoT devices with limited power budget. Figure 1 shows a high-level architecture of NVDLA. As it can be seen in this figure, NVDLA has three major top-level blocks. Convolutional Core is consisted of multiply-accumulate (MAC) units for matrix-matrix multiplication in convolutional and fully-connected layers of a DNN. The input activations and filter weights are stored in Convolutional Buffer which are then fed into Convolutional Core. Post-processing unit is comprised of sub-units which perform various processes such as pooling and applying non-linear activation functions. These three blocks are programmed and controlled by Configuration and Control Block, which is accessed by the host processor via Configuration and Space Bus (CSB) interface. All the processing units are connected to Memory Interface block. This block arbitrates the access to main memory via Data Backbone (DBB) interface. The CSB and DBB interfaces are further described in Section 3.

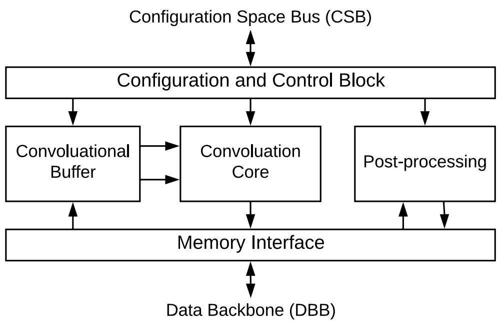


Figure 1. NVDLA architecture. Adopted from [14].

2.2 FireSim

FireSim is a fast cycle-exact system simulator which runs on cloud FPGA. In FireSim, the simulated hardware is derived from the actual RTL of the design, thus, the same RTL can be pushed through a VLSI flow to fabricate the chip, and since the simulation is running on FPGA, it is orders of magnitude faster than software-based simulation. In addition, FireSim can be used for debug and trace generation. Lastly, since it is running on the cloud, there is no need to pay the upfront cost to buy FPGA boards. These traits make FireSim more preferable in many aspects comparing to software-based simulators to be used in areas of computer architecture and system research. For instance, it provides the capabilities of faster and more accurate performance analysis and true hardware/software co-design.

3 NVDLA Integration

For this integration, we choose the *nv_large* configuration of NVDLA which has 2048 MAC units and a 512 KiB convolutional buffer. NVDLA connects to the rest of the SoC via these interfaces:

Configuration Space Bus (CSB): A low-bandwidth slave interface which provides access for the host processor to configure NVDLA and read the status. The CSB interface implements a simple custom protocol. An adapter is supplied in the NVDLA code repository to convert this simple protocol to ARM Advanced Peripheral Bus (APB) [2].

Data Backbone (DBB): A high-bandwidth master interface which is connected to the memory system. NVDLA uses this interface to access the main memory. DBB implements ARM Advanced eXtensible Interface (AXI) [3] protocol.

IRQ: A 1-bit interrupt line which is asserted when NVDLA finishes an on-going task or an error occurs.

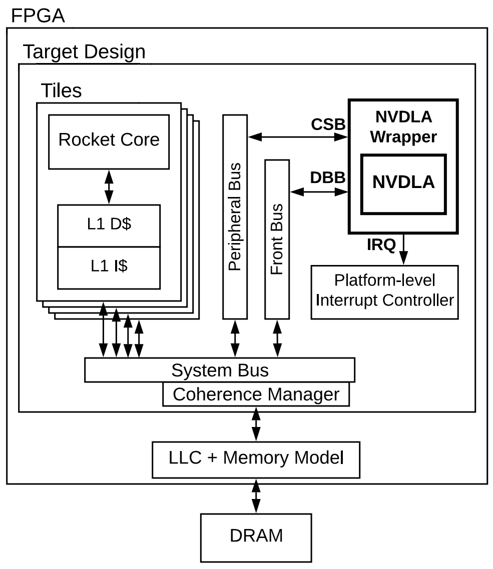


Figure 2. Simulator overview with our NVDLA integration.

FireSim's target design is derived from Rocket Chip SoC generator [4], which is written in Chisel hardware construction language [5] and uses TileLink bus protocol [17] for on-chip communication. Figure 2 shows the overview of the simulator with our NVDLA integration. We describe the blocks in this figure which are important for understanding our integration:

Front Bus: A TileLink switch that arbitrates non-CPU masters and connects them to the memory system. NVDLA is the only non-CPU master in our design, therefore, Front Bus is only connected to DBB interface of NVDLA.

Peripheral Bus: A TileLink switch that connects the slave devices and maps them to a memory region. We connect CSB interface to Peripheral Bus in order for the processor to access the memory-mapped registers of NVDLA. To save space, the other peripherals connected to this switch are not shown in Figure 2.

Platform-level Interrupt Controller (PLIC): A prioritized interrupt controller which routes the interrupt sources of devices to CPU cores. We connect interrupt line of NVDLA and other devices (not shown in Figure 2) to PLIC.

NVDLA Wrapper: Since Rocket Chip uses TileLink bus protocol for on-chip communication, we create this wrapper layer that converts bus transactions between the NVDLA (which uses ARM APB and AXI bus protocols) and the Rocket Chip SoC.

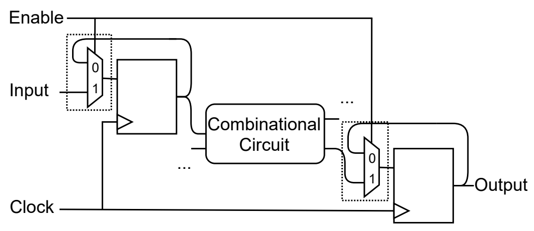
LLC and Memory Model: This model is not part of the Rocket Chip SoC and is added to the simulation environment by FireSim to model accurate LLC and main memory timing. Note that this model is written in Chisel and is implemented on FPGA to avoid being a bottleneck for the simulation speed. The FireSim user can choose the exact behavior from a variety of models such as an ideal memory with constant access latency or a DRAM model. The LLC model is configurable at the runtime. The number of sets, ways, and block size can be modified without having to rebuild the FPGA image. We use this feature to analyze the performance of NVDLA for multiple different number of sets and block sizes in Subsection 4.1.

Thanks to the fast integration and development capabilities that Chisel offers, we find it convenient to reuse the SiFive's NVDLA integration code for our purpose. We still need to create the NVDLA simulation model and modify the FireSim and Amazon Web Services (AWS) FPGA flow to integrate NVDLA on FireSim. We describe our method of creating the NVDLA simulation model in the following.

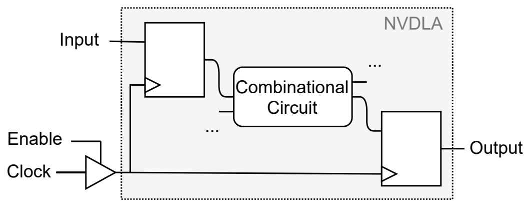
3.1 Creating the NVDLA Simulation Model

FireSim uses FAME-1 transform [11, 20] to translate the target RTL to the target model and create a token-based simulator. In each target cycle, the target model reads a token on its input and generates a token on the output. However, there can be cycles in which a token is not available to be consumed by the target model. For instance, the memory model can be configured to service the memory accesses in 10 cycles while it may take longer for the FPGA DRAM to actually read the data in order for the memory model to pass it to the target model. In this case, FireSim stalls the target model until the FPGA DRAM reads the data and a token is available to be consumed by the target model.

In order to stall the target model, FireSim runs a pass in the back-end of Chisel compiler, which adds a global enable signal and an input mux for each register in the target design as shown in Figure 3a. By deasserting the enable signal, the target model is stalled at the cycles which a token is not available on the input. Since NVDLA source is written in Verilog, the pass cannot be directly applied. Instead, we add a new pass in the Chisel back-end to identify the Verilog modules in the design and apply the FAME-1 transform by clock-gating as shown in Figure 3b. This new pass identifies the input clock source of the NVDLA Verilog module and adds a clock buffer with an enable signal. Once the enable signal is deasserted, the clock input remains low to stall the NVDLA model.



(a) Register enable.



(b) Clock-gating.

Figure 3. FAME-1 transform with (a) register enable (b) clock-gating.

4 Performance Analysis

In this section, we demonstrate the capabilities of our integrated platform for performance analysis. Specifically, we analyze the effect of interference between the NVDLA and RISC-V CPU cores in accessing the shared memory hierarchy including LLC and DRAM.

For this purpose, we choose the state-of-the-art YOLOv3 object detection algorithm [16] as a benchmark. YOLOv3 incorporates a DNN that needs 66 billion operations to process a 416×416 frame. Table 1 shows our platform configuration parameters. Note that NVDLA block is clocked at the same frequency with the processor, due to a limitation of current FireSim implementation, which requires all hardware blocks to have synchronous clock sources. In a real chip, NVDLA may be operated at a lower frequency. We hope that this will be fixed in near future as FireSim is planned to support having asynchronous clocks.

On this configuration, we first measure the baseline performance of NVDLA. The time that it takes to process a frame on this platform is 133 ms out of which 67 ms is spent on NVDLA and the rest on the processor. Some of the computations which are not supported by NVDLA are executed by

Table 1. Baseline platform configuration.

Processor	Quad-core, in-order, single-issue, 3.2 GHz
NVDLA	2048 INT8 MACs, 512 KiB buffer, 3.2 GHz
L1 I/D$	Private 16/16 KiB, 4-way, 64 B block
LLC	Shared 2 MiB, 8-way, 64 B block
DRAM	16 GiB DDR3, 4 ranks, 8 banks, FR-FCFS

the processor. These are particularly upsampling, floating-point to integer conversion (and vice-versa), and custom YOLO layers. We optimized these layers with OpenMP to utilize all four processor cores.

To calculate the speedup achieved by using NVDLA, we measure the performance of YOLOv3 when Rocket Cores are performing all the computations. In addition, we run the benchmark on two other platforms: 1) Intel Xeon E5-2658 v3 CPU (2 sockets; 24 cores/48 threads in total) and 2) Nvidia Titan Xp GPU. The benchmark is running multithreaded on Rocket Core and Xeon platforms. Note that computations are performed with 8-bit integer precision on NVDLA and with single-precision floating-point on the rest of the platforms. We use the open-source Darknet [15] neural network framework to conduct our experiments.

Figure 4 shows the number of frames per second (FPS) on each platform. It can be seen that using NVDLA, the performance is improved by 407x comparing to the case, which Rocket Cores are performing all the computations. In this experiment, Titan Xp can extract the highest level of parallelism and achieves the FPS of 41 thanks to its 3840 CUDA cores. This is 5.5 times faster than NVDLA, meanwhile, according to power and area numbers published on [12] and [1], NVDLA consumes much less power and chip area than Titan Xp and, therefore, can provide the capability of running computationally-intensive tasks in real-time on low-cost platforms with limited power and size.

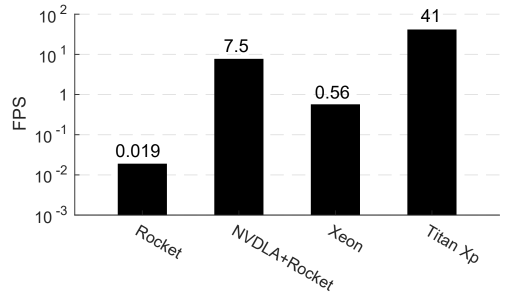


Figure 4. The performance of YOLOv3 object-detection algorithm on different platforms.

4.1 Effect of Last-level Cache on Performance

Sharing the LLC with the DNN accelerator has some benefits comparing to using a scratch pad. First, chip area can be saved by removing a large SRAM which implements the scratch pad. Second, less programming effort is needed as the data movement is managed by the cache controller. In this experiment, we examine the performance improvement achieved by sharing the LLC. We vary the LLC size and measure NVDLA speedup with respect to a design with no LLC. The LLC size is changed by varying the number of cache sets. For each LLC size, we also varied the size of cache blocks (32B, 64B and 128B) to see their impact to the speedup. The rest of the parameters are the same as the baseline configuration in Table 1. Since our focus is on the performance of NVDLA, the speedup is measured for the layers of YOLOv3 that run on NVDLA.

Figure 5 shows the results. First, let us focus on the baseline '64B blocks' results. For 64-byte cache blocks configuration, adding an LLC considerably improves performance (up to 1.28x speedup). Interestingly, however, the size of the LLC does not have much differences. Specifically, when the LLC size is at 0.5 KiB, it achieves 1.17x speedup, which is not much different from the maximum speedup of 1.28x, which is achieved at the LLC size of 64 KiB.

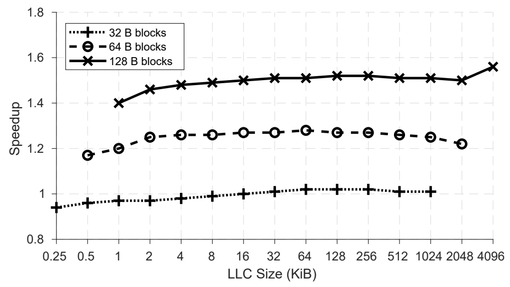


Figure 5. NVDLA speedup achieved by using the last-level cache.

As shown above the performance of NVDLA is not very sensitive to the LLC size. This is due to the large convolutional buffer that captures most of the temporal locality in NVDLA memory accesses. However, the results in Figure 5 shows that NVDLA performance varies to a much larger extent when the cache block size is varied. For instance, the speedup achieved by using LLCs with the constant capacity of 1024 KiB and different block sizes of 32B, 64B, and 128B is 1.01x, 1.25, and 1.51x respectively. The maximum speedup which can be achieved by using the LLC is 1.56x for a 4096 KiB cache with 128-byte blocks. The reason is that the minimum burst size of NVLDA memory accesses is 32 bytes, therefore, having cache blocks larger than 32 bytes helps to reduce the latency for larger bursts or later accesses to nearby memory locations. This shows most of the benefit of sharing the LLC comes from capturing the spatial locality in NVDLA memory references. Therefore, it is likely that hardware prefetching further improves NVDLA performance on this platform as it can bring useful data into the cache before it is accessed by NVDLA.

4.2 Effect of Shared Memory Interference

Since NVDLA and the CPU are sharing the memory system, it is likely that they interfere with one another when accessing the memory. This in turn can result in unpredictable latency in execution of the tasks running on NVDLA. This problem becomes more important on real-time systems in which the total correctness of operation depends on both the logical correctness of computation and the time that it is performed.

We use Bandwidth Write (BwWrite) benchmark [21] to study the interference caused by the tasks running on the processor. BwWrite is a synthetic benchmark which writes data to sequential memory addresses to generate the maximum memory traffic by the processor. Is it possible to set the working set size (WSS) of BwWrite to target different levels of the memory hierarchy. We co-schedule BwWrite with YOLOv3 running on NVDLA and vary the set size of BwWrite to either fit into L1 cache, LLC, or DRAM. The number of co-scheduled BwWrites is also varied from 1 to 4. We pinned BwWrites to specific cores by setting the CPU affinity in Linux. The execution time on NVDLA is measured and normalized to *solo* execution time i.e. NVDLA running in isolation with no BwWrite running on the cores.

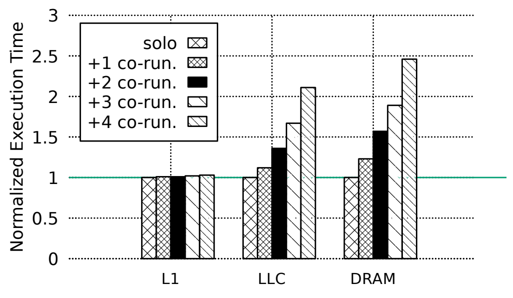


Figure 6. Normalized execution time of NVDLA. The x-axis denotes the working-set size of BwWrite co-runners.

Figure 6 shows that there is no slowdown when the working set of BwWrite fits into the L1 cache. The reason is each core has its own private data cache and as long as the WSS is smaller than L1, there is no access to the shared memory. When the WSS of BwWrite is LLC-fitting and is larger than L1 cache size (denoted with *LLC* in the figure), it takes longer for NVDLA to finish the job as the memory accesses are delayed due the concurrent accesses by the cores to the shared bus and the LLC. This can results in up to 2.1x slowdown when four BwWrites are running in parallel. The slowdown increases to 2.5x for 4 co-runners when WSS is DRAM-fitting since interference in the DRAM controller scheduler queue and the DRAM banks adds more delay to NVDLA memory access time.

5 Conclusion

In this paper, we described our integration of NVDLA into a RISC-V multicore SoC on FireSim. The integrated SoC platform runs on Amazon cloud and does not require any physical FPGA board. Furthermore, thanks to FireSim, the performance critical shared cache and memory subsystems are easily configurable. Therefore, we believe that our platform is a flexible and cost-effective solution for conducting research. As a case study, we ported YOLOv3 and evaluated its acceleration performance, compared to GPU and CPU solutions, and under various cache and memory configurations. We find that (1) NVDLA provides good acceleration performance, especially considering its low power consumption, (2) larger cache block size and/or hardware prefetcher is desirable for performance, and (3) the impact of shared memory interference between CPU and NVDLA is significant—which can be especially problematic for critical real-time embedded systems found in automotive applications—suggesting the need of additional QoS mechanisms (e.g., [6, 8, 9]).

Acknowledgments

This research is supported in part by NSF grants CNS 1718880, CNS 1815959, and NSA Science of Security initiative contract no. #H98230-18-D-0009.

References

[1] [n. d.]. NVIDIA Titan Xp. https://goo.gl/jWvXqv
[2] Arm. 2004. AMBA 3 APB Protocol v1.0 Specification.
[3] Arm. 2013. AMBA AXI and ACE Protocol Specification.
[4] Krste Asanović et al. 2016. *The Rocket Chip Generator.* Technical Report. EECS Department, University of California, Berkeley.
[5] Jonathan Bachrach et al. 2012. Chisel: Constructing hardware in a Scala embedded language. In *DAC*. IEEE, 1212–1221.
[6] Farzad Farshchi et al. 2018. Deterministic memory abstraction and supporting multicore system architecture. In *ECRTS*, Vol. 106.
[7] Mingyu Gao et al. 2017. Tetris: Scalable and efficient neural network acceleration with 3D memory. *OSR* 51, 2 (2017), 751–764.
[8] Boris Grot et al. 2012. A QoS-enabled on-die interconnect fabric for kilo-node chips. *Micro* 32, 3 (2012), 17–25.
[9] Ravi Iyer et al. 2007. QoS policies and architecture for cache/memory in CMP platforms. In *PER*, Vol. 35. ACM, 25–36.
[10] Sagar Karandikar et al. 2018. Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *ISCA*. 29–42.
[11] Donggyu Kim et al. 2016. Strober: Fast and accurate sample-based energy simulation for arbitrary RTL. In *ISCA*. IEEE, 128–139.
[12] Nvidia. 2018. NVIDIA Deep Learning Accelerator. http://nvdla.org/primer.html
[13] Nvidia. 2018. NVIDIA Deep Learning Accelerator IP to be Integrated into Arm Project Trillium Platform. https://goo.gl/o8rfff
[14] Nvidia. 2018. The Nvidia Deep Learning Accelerator. https://goo.gl/Znyba5
[15] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.
[16] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
[17] SiFive. 2017. SiFive TileLink Specification.
[18] SiFive. 2018. First Open-Source RISC-V-Based SoC Platform with Nvidia Deep Learning Accelerator Technology. https://goo.gl/rdPSbJ
[19] SiFive. 2018. HiFive Unleashed. https://goo.gl/TZtno2
[20] Zhangxi Tan et al. 2010. A case for FAME: FPGA architecture model execution. In *CAN*, Vol. 38. ACM, 290–301.
[21] Prathap Kumar Valsan et al. 2016. Taming non-blocking caches to improve isolation in multicore real-time systems. In *RTAS*. IEEE, 1–12.
[22] Xilinx. [n. d.]. Xilinx Virtex UltraScale+ FPGA VCU118 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/vcu118.html